Spring 5-22-2019

Learning To Play The Trading Game

Neeraj Kulkarni
San Jose State University

Learning To Play The Trading Game

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Neeraj Kulkarni

May 2019

The Designated Project Committee Approves the Project Titled

Learning To Play The Trading Game

by

Neeraj Kulkarni

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2019

Dr. Katerina Potika	Department of Computer Science
Dr. Sami Khuri	Department of Computer Science
Dr. Mike Wu	Department of Computer Science

ABSTRACT

Learning To Play The Trading Game

by Neeraj Kulkarni

Can we train a stock trading bot that can take decisions in high-entropy environments like stock markets to generate profits based on some optimal policy? Can we further extend this learning for any general trading problem? Quantitative Algorithms are responsible for more than 75% of the stock trading around the world. Creating a stock market prediction model is comparatively easy. But creating a profitable prediction model is still considered as a challenging task in the field of machine learning and deep learning due to the unpredictability of the financial markets. Using biologically inspired computing techniques of reinforcement learning (RL) and artificial neural networks(ANN), this project attempts to train an agent who takes decisions based on the optimal decision policies learned. Different existing RL techniques and their slightly modified variants will be used to train the agent, and the trained model is then tested against different stock prices and also stock portfolio settings to see if the agent has learned the rules of the game and can it act optimally irrespective of the testing data provided. This work aims to provide general users with simple recommendations about the possible investment decisions of selected stocks in the portfolio. Results of the implemented approaches do seem to work somewhat well on specific periods of stock market time series, but they are observed to be fragile. Selected strategies do not guarantee similar results on all historical time-periods, nor they are guaranteed to provide exceptional performance on unpredictable future stock market time-series data.

Index Terms - **Reinforcement Learning, Artificial Neural Networks, Deep Learning, Recurrent Neural Networks, Long Short-Term Memory,**

Time Series Analysis, Deep Q-learning, Direct Reinforcement Learning.

ACKNOWLEDGMENTS

I want to thank my project advisor Dr. Katerina Potika for her continued support and guidance throughout this master's project work. Her expertise helped me work efficiently on complex project topics and learn the basic concepts in a limited amount of time. I would also like to thank Dr. Sami Khuri and Dr. Mike Wu for accepting to be a part of my project defense committee. I appreciate the efforts taken by my advisor and the other committee members in helping me complete this work. Their suggestions, review comments and feedback have helped me refine my work over the last two semesters. Their valuable inputs have helped me steer the project in the right direction.

Finally, I would like to thank my mother and my friends for their continued support. Without their help, I would not have been able to complete this work.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Stock Market 101

If someone owns a stock of a specific company, they own a piece of that company. As and when the companies need capital for growth, they sell part of the ownership of the company in the form of stocks to the investors. Based on the performance and prospects, investors buy shares. If the company outperforms the expectations of the investors, then the value of the stock goes up. Investors vie for the limited amount the stocks of such companies out in the stock market. Similarly, if something goes wrong and the company posts below expectation performance results, the stock price for that particular company goes down, and more and more investors try to divest and prevent the losses from such bad investments.

Stock markets are the virtual marketplaces where people buy and sell their stocks. Investors and traders use these stock price ups and downs to generate profits. The simple rule for profit maximization in this stock trading game is - Buy Low and Sell High. Even though it sounds straightforward, the movement of stock prices is not that easy to predict. This time-series data is considered a random walk model which states that price changes in the stock markets are erratic. One cannot predict future stock prices with absolute conviction just using historical data [1] [2]. An author and a Princeton University Economics professor, Burton Malkiel even claimed that "a blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts." [3]. Apart from theoretical models of random walk for stock prices, due to the dynamic nature (the volatility) of stock prices and uncertainty surrounding the stock market, stock

trading is considered an act of gambling. And hence, very few individuals engage in the stock market trading and stock market investments carry a notion of high-risk investment.

Few scientists and researchers [4] oppose the random walk theory and claim that historical prices can provide some patterns and insights related to the current and future stock price estimates and stock movement trend. Even though the accuracy of systems that use such models for future price prediction is not that high, these patterns do help investors make an informed decision about future actions.

The following subsection introduces how different prediction systems try to find patterns in the data and why Artificial Intelligence (AI) techniques and reinforcement learning can be used for this specific task.

1.2 Stock Price Prediction

There has been a lot of buzz around Artificial Intelligence (AI) in recent years, and the applications of AI related to various domains have exploded in the same period. Various machine learning (ML) and Deep learning (DL) techniques have been successfully applied to a wide array of problems ranging from image processing, natural language processing to time series analysis. The simple goal of ML and DL algorithms is to either learn some patterns in the given data or to find the relation between the given data and given outputs so that this model can predict the correct output when some unseen sample data is fed to the system in the future. Mostly complex mathematical models are used to approximate the relationship between inputs and output labels or patterns in the input data.

One such area where different AI techniques are heavily applied is stock markets. Since the invention of computing machines, it is believed that computers can crunch

numbers better than human brains. As stock market prediction is a lucrative field, billions of dollars are invested in this research every year. Many statistical and machine learning techniques have been developed and applied to stock price prediction since the last three decades with varying degrees of success. There is no 100% accurate prediction algorithm (at least publicly known) which accurately predicts the future price of every stock in the portfolio. In the last few years, Artificial neural networks (ANN) have been particularly successful in modeling time series data using recurrent network architecture. The performance of ANNs mainly depends on the amount of data and available computational power. If a large dataset is fed to ANNs, they refine and tune their performance using learn and adapt strategies on multiple examples. One complication with ANNs is that they require extensive time for training, and they tend to overfit the training data. As stock market time-series data have high movement and noise, a model on a specific period might not generalize well on the unseen data. Due to the uncertainty and scale of data, training offline neural networks on stock market data leads to approximate and less accurate models. Also, as mentioned in the previous subsection, historical stock prices alone cannot accurately forecast future prices. Numerous external stochastic factors influence stock prices. Prediction models need to consider such factors to improve accuracy. Adding additional parameters even further increases the model complexity and training time.

The best way to make the decisions regarding buying, selling or holding (neutral) any stocks in a volatile manner should be taken in an interactive manner to reap significant profits and avoid sudden losses due to the uncertainty of the environment. A subset of machine intelligence which is heavily influenced by biologically inspired computing named Reinforcement learning (RL) plays a vital role in online machine learning and learning decision policies. RL is the closest the way humans and animals

learn by interacting with the environment. This project aims to use various techniques of Deep learning and Reinforcement learning to generate optimal policy which helps the agent to make decisions regarding stocks in a dynamic and volatile environment to generate profits without violating the rules of this environment.

The organization of this project report is as follows. Chapter 2 states the problem statement and objectives of this project work. Chapter 3 lays the foundation for the work presented in this project by briefly explaining core terminologies and algorithms used in the implementation, experimentation and analysis parts of this work. Chapter 4 reviews previous work related to the use of deep learning and reinforcement learning for the stock market prediction problem and intelligent stock trading agent development. Chapter 5 provides information about the data used, its corresponding features and various data preprocessing techniques used in the experimentation phase of this project. Chapter 6 is the experimentation section that provides results and analysis of multiple experiments performed while training this stock trading agent. The following chapter, chapter 7, provides a summary of the work done throughout this project. And, then the final section concludes this project report by providing future directions.

CHAPTER 2

Problem Definition

The main objectives of this project can be summed up in a set of questions related to different categories as follows:

- General Objective
  - Can we train an agent who will be able to generate profits in the volatile and highly unpredictable financial stock markets based on some optimal policy learned through training?
- Performance Objectives
  - Can we train a model which will be able to learn a decision policy for the stock market time-series data or portfolio of stocks instead of just predicting the trend of the series
  - How well will the agent perform if we feed financial time series data for learning? Will it generate profit only using information about (theoretically) unpredictable time series?
  - How various reinforcement learning techniques perform for developing an optimal trading strategy? How well these models behave on multiple time windows in stock price time-series with different characteristics (like trending market, volatile market, etc.)?
  - If the agent learns an optimal decision strategy then can this strategy be applied to the other stocks and other commodities without retraining the model on the test price time-series of the selected stock or commodity?

– How well the agent learns an optimal decision strategy when we feed additional features like technical indicators along with historical price data?

CHAPTER 3

Core Concepts and Terminologies

3.1 Reinforcement Learning

Reinforcement learning (RL) is a field of Artificial intelligence that involves goal-oriented learning through interaction with the environment. The learner interacts with the environment and gets some response or reward along with the next state from the environment. The environment in which the learner is operating might transition to some other state due to the actions performed by the learner. Then for the next time-step, the learner/agent needs to take appropriate action based on the changed environment state to maximize the reward. Figure 1 shows simple environment interaction and feedback loop for the agent.

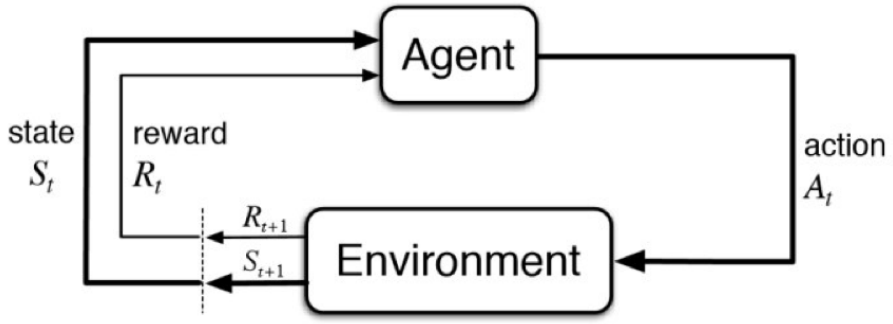


Figure 1: Reinforcement Learning: Iterative Learning Loop.
Source: R. Sutton and A. Barto,Reinforcement learning, an introduction.

In an unknown and unseen environment, rather than generalizing the solution using a supervised machine learning approach, the learner should be able to learn through interactions with the environment.

Two critical features of reinforcement learning are a trial-and-error approach and delayed rewards [5]. The trial-and-error approach presents the challenge of

exploration-exploitation trade-off in reinforcement learning. The agent needs to exploit the past experiences which provided effective results to maximize the future reward. But it also simultaneously needs to explore new actions that might generate even greater rewards. The exploration and exploitation process can only be pursued through failures. This problem of the trade-off between exploration-exploitation in the field of reinforcement learning and automatic control is still an open research issue. The other feature of delayed rewards states that the agent must take actions not only based on immediate rewards but also keeping long term future rewards in mind. This chapter provides a brief introduction of simple reinforcement learning algorithms. Before we start with the actual algorithms, the following subsection provides details about required notations and definitions used in RL algorithms.

3.1.1 Notations and definitions

- State: $s \in S$, a state in the possible state space. For the stock-trading problem, state space is huge.

- Action: $a \in A$, action in the possible action space. The action space for this problem is discrete and to be precise it is $\{Buy, Hold, Sell\}$

- Policy: π: The core idea of reinforcement learning in which agent decides which action to take based on the current state in which the agent is in.

  - Deterministic Policy: $\pi(s) = a$, This policy clearly states which action to take when the agent is in the current state s. Q-learning algorithm is a deterministic policy algorithm which outputs action which takes the agent to the state with maximum possible next Q-value state from current state

  - Stochastic Policy - $\pi(a|s) = (0, 1]$ This policy gives probability distribu-

tion over all possible actions given current state. Policy gradient algorithms are stochastic policy algorithms

- Reward $r(s, a)$: Reward that agent gets when in the state s, it takes action a. Rewards are mostly scalar values, and they represent the goal or objectives of the agents.

- Discount Factor $\gamma \in [0, 1]$: This entity decides if the agent cares more about short term rewards or long term rewards. Larger the gamma, far-sighted is your agent, and smaller discount factor entails agent with a myopic view. Financial time series problems mostly care about short term rewards; hence the discount factor values in such settings are usually smaller.

- Return $R^{\pi} = \mathbb{E}[\sum_{t}^{T} \gamma^{T} r(s_t, a_t)]$: Expected discounted future reward accumulated over time period T starting from current time t.

- Learning Methods

  - Model-based learning methods: Model based learning methods try to learn the transition functions $T(s, a)$ as well as reward function $r(s, a)$. But, it requires internal details of the environment like transition probabilities from one state to other states reachable from the current state, $\mathbb{P}(s^{'}|s, a)$ which are not available in most of the real world environments.

  - Model-free learning methods: In most of the real world environments, internal details like state transition probabilities of the environment are not know the outside agent. Hence, the agent cannot learn transition function and has to learn just by sampling the state-action pairs and recording rewards. Then the agent tries to fit states and actions pairs using approximators so that it generalized well for future unseen states.

* Q-learning: Learn Q-Function $F\langle s, a\rangle \rightarrow Q(s, a)$. The agent then takes action from the current state based on the maximum Q-value next state rather than depending on transition probabilities, unlike model-based learning methods.
* Policy Gradients: Agent learns the Mapping $F : s \rightarrow a$ without estimating transition rewards from one state to another state given particular action, i.e. no estimation of state and action values functions.

- Value functions: Value functions determine goodness a specific state in the long run. There is a subtle difference between the reward and the value function of a state. the reward $r(s, a)$ defines immediate gains or losses if the agent takes some action a in state s, whereas value function is a long-term value assessment of a specific state considering expected discounted future rewards. They are roughly classified into the following two classes.
  - State Value functions: Expected value of the state s if the agent follows behaviour policy π

$$V^{\pi}(s) = \mathbb{E}[\sum_{t} \gamma^{T} r_t | s_t = s]$$

  - Action Value functions: Expected value or quality of the given state-action pair ($s,\ a$), if the agent follows given behavioural policy π

$$Q^{\pi}(s, a) = \mathbb{E}[\sum_{t} \gamma^{T} r_t | s_t = s, a_t = a]$$

(Note: Referred from notes of [6])

3.1.2 Q-learning

As mentioned above, Q-learning [7] is model-free, off-policy learning algorithm for reinforcement learning. Q-learning tries to learn optimal policy

$Q^{\pi^*}(s,a) = max_\pi Q^{\pi^*}(s,a), \forall s, a$. In most of the cases transition probabilities of the environment are not fully known to the agent, hence the Q-value of a state, action pair is computed using action value functions. The Q-value of a state-action pair is calculated as follows,

$$Q^\pi(s,a) = \mathbb{E}_\pi[r_t + \gamma max_{a'} Q(s^{'}, a^{'})]$$

This Q-value for state-action pair can be computed using recursive Bellman equations iteratively to learn the optimal Q policy.

This Q-learning algorithm is an off-policy reinforcement learning algorithm where we have two policies namely π and μ. The agent takes actions as per behavior policy μ. The end objective for the agent is to learn the target policy π from experiences sampled from μ. From the term, $max_{a'} Q(s^{'}, a^{'})$ in the Q-value computation, it is clear that agent acts greedily with respect to target policy. Whereas the behaviour policy μ is ϵ - greedy with respect to $Q(s,a)$.

ϵ - greedy behavior is widely used policy behavior in reinforcement learning, and it mainly helps with the dilemma of exploration-exploitation. In this policy, the agent picks uniformly random actions with probability ϵ (exploration) and with probability 1 - ϵ, the agent plays greedily (exploitation). Rather than keeping a fixed ϵ, an improved mechanism of exponential ϵ decay is used as shown in Figure 2. In the beginning, when the agent is new to the environment, the agent explores with more probability. But as time progresses and agent plays more episodes, instead of taking random actions, the reduced ϵ probability forces the agent to exploit the learning and take steps greedily.

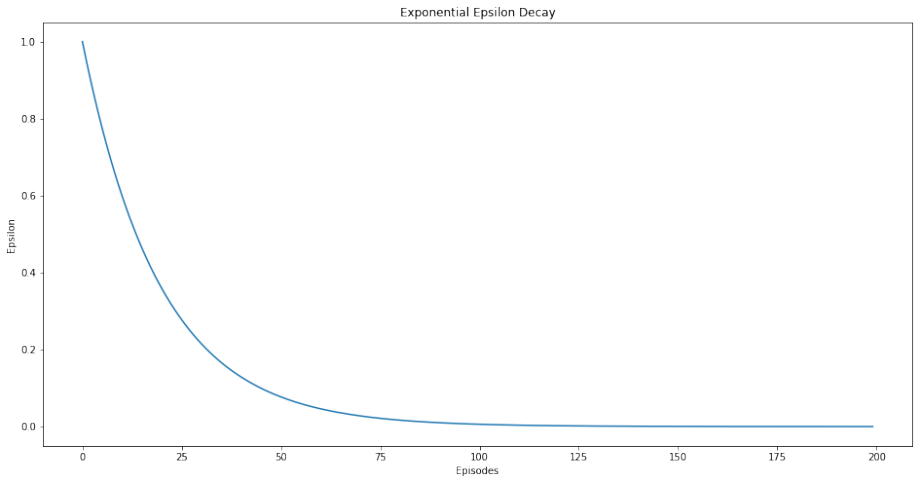


Figure 2: Epsilon decay mechanism.

Algorithm 1: Q-learning

1 Initialize $\hat{Q}(s, a)$ arbitrarily $\forall s, a$;
2 Observe initial state $s = s_0$ **repeat**

1. Take an action a based on behaviour policy μ.
2. Observe the reward r and next state $s^{'}$ given by the environment.
3. update $\hat{Q}(s, a) = \hat{Q}(s, a) + \alpha[r + \gamma max_{a'} Q(s^{'}, a^{'}) - \hat{Q}(s, a)]$

3 **until** *convergence*;
4 Optimal Policy: $\pi^*(s) = max_a \hat{Q}(s, a)$

3.1.3 Deep Q-learning

Calculation of Q-values in a continuous and large state space faces the problem of Bellman's curse of dimensionality. Computing Q-values for all possible combinations of state-action pairs and storing them in the Q-table becomes computationally intractable. Agents in most of the real world settings don't even traverse whole state-action spaces in their respective environments. Instead, they traverse specific state paths more frequently. Hence, instead of computing the Q-values for all possible state-action pairs, function approximators are used which sample the state paths from

state space and then try to fit the a value function for future expected return values for these paths. Function approximators try to fit the Q-function to the state-action pairs that the agent has seen and generalize this function to the future states or unseen states. The use of deep neural networks as function approximators can help Q-networks learn a non-linear mapping between state-action pairs and associated expected returns. They try to minimize the loss function.

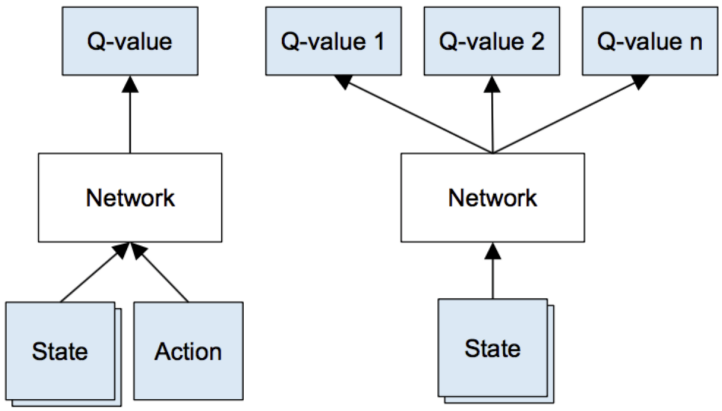


Figure 3: *Left:* Naive formulation of deep Q-network. *Right:* More optimized architecture of deep Q-network, used in DeepMind paper.

Figure 3: Deep Q-Network Architecture.
Source: https://www.intel.ai/demystifying-deep-reinforcement-learning/

3.1.4 Improvements to Deep Q-networks

Various Deep Q-learning improvement mechanisms like experience replay and fixed Q-targets were also implemented to help the deep Q-learning algorithm converge better and faster.

Experience replay mechanism stores the samples of $\langle s, a, r, s' \rangle$ in the replay memory and then we apply supervised learning algorithms to try and fit behavior network to target network.

Algorithm 2: Deep Q-learning

Initialize replay memory R
Initialize batch size to the required minibatch size number
Initialize Total Episodes = The total number of episodes for the agent to play the game.
Initialize behaviour Policy $\mu = \hat{Q}(s, a)$ arbitrarily $\forall s, a$;
Initialize target policy $\pi = Q(s, a) = \hat{Q}(s, a), \forall s, a$;
Observe initial state $s = s_0$
repeat
    Take an action a based on behaviour policy μ.
    Observe the reward r and next state $s^{'}$ given by or received from the environment.
    store $\langle s, a, r, s^{'} \rangle$ in replay memory where $s = s_t$ and $s^{'} = s_{t+1}$
    Shuffle experience in the replay memory and sample random samples $\langle s, a, r, s^{'} \rangle$ from replay memory and create mini-batch of samples
    for *all samples in the mini-batch* **do**
        Compute Target Q-value of state s using target policy neural network π as $r + \gamma max_{a^{'}} Q(s^{'}, a^{'}; \theta_i^t)$
        Compute Current Q-value, $Q(s, a; \theta_i)$ of state s using behaviour policy μ
        Compute the loss as $\mathcal{L}(s, a) = (r + \gamma max_{a^{'}} Q(s^{'}, a^{'}) - Q(s, a))^2$
        Fit the behavior neural network μ to minimize the loss $\mathcal{L}(s, a)$
    endFor
until *(Current Episode < Total Episodes) or convergence*;
Optimal Policy: $\pi^*(s, \theta) = max_a \hat{Q}(s, a; \theta)$

When we use the same neural network for the current Q-value estimation (Q-estimation) and target Q-value (Q-target) prediction for the next state, it becomes hard for the Q-estimation network to fit to the Q-target network as both networks (virtually the same network) move after each training iteration with updates to the network weights. Hence, a simple solution is suggested in [8], where the q-target network is kept fixed for a certain number of episodes, and we try to fit q-estimation network as accurately as possible to this fixed q-target network and then after a certain number of episodes we copy the weights of q-estimation network to the Q-target network. This mechanism resolves the problem of moving target network and

helps the Q-estimation network to converge much faster.

3.1.5 Policy Gradients and Direct reinforcement learning

Instead of estimating the state or action value function and then taking action greedily or with some exploration-exploitation policy like ϵ - greedy policy, policy gradient methods directly try to parameterize the policy and learn stochastic policies where an agent learns a probability distribution over the action space given a specific state. This probability distribution then converges to the optimum policy π with most probable action in a particular state slowly converging to 1.

The parametric model for policy in policy gradients often involves the use of the Neural network to fit the decision policy to the action probability distribution given the states of the environment. Policy gradients do not need to forecast anything to generate expected future rewards in the future which are very crucial part of Q-learning and more precisely deep Q-learning algorithm. In unpredictable environment, these methods helps agent focus more on immediate returns and avoid making guesses about expected future rewards.

In simple words, the flow of policy gradient training (Actor-critic method and direct reinforcement learning methods to be precise) is as follows - Play the whole episode, take actions sampled using soft-max probability suggested by the neural network (actor). Using the same network with different last layer, calculate state value (only for actor-critic methods and not for direct reinforcement learning which is just actor based method) using state value function (critic) to find out the goodness of that action in that specific state. update policy using gradient ascent at the end of the episode using the samples stored in the memory

Direct reinforcement learning algorithm by Moody et.al. [9], suggest using just

actor based methods with the differentiable objective function. They also suggest profit rewards achieved when we take that specific action to be used as the value for that state-action pair. Policies are updates directly using observed profit as the reward. For our experimentation, we will be using a slightly modified version of [10] which in turn use the core idea of [9].

Details about the setup of policy gradient experiments are provided in Chapter 6.

3.2 Artificial Neural Networks

Artificial Neural Networks are a branch of machine intelligence, which are based on the concept of a network of neurons in the human brain. This architecture allows ANNs to learn complex tasks like image, speech, audio and pattern recognition using the process of learning and adapting through experiences closely analogous to the human learning process. Simple neural network architecture is stacked layers of neurons with layers classified into three categories namely, input, output, and one or more hidden layers.

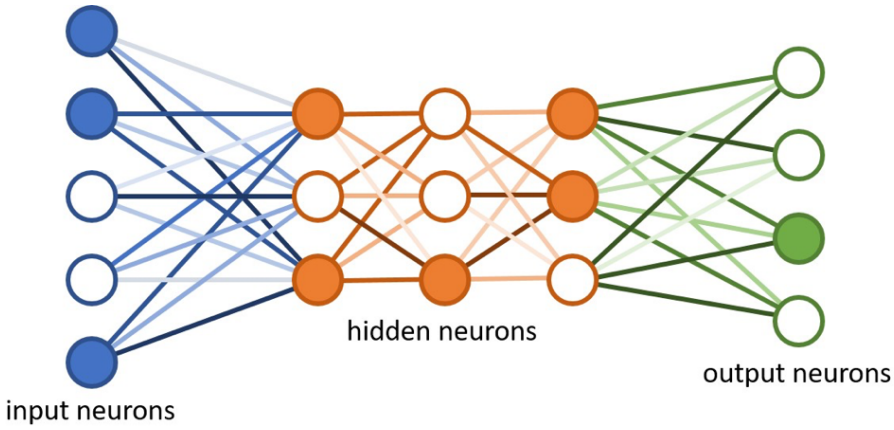


Figure 4: Simple Artificial Neural Network Architecture.
Source: https://medium.com/xanaduai/making-a-neural-network-quantum

There are multiple types of ANNs, but most common architectures include simple

feed-forward network, recurrent neural networks (RNNs) and convolutional neural networks (CNNs) [11] [12]. Feed-forward CNNs are suitable for applications involving images and videos whereas RNNs are used in the applications with sequential inputs such as text, speech, audio, language and time series data [11] [12]. As for this project we are concerned with time series data analysis, our primary focus will be simple feed-forward neural networks, recurrent neural networks and different variations of the recurrent architecture. Following subsections elaborate details of these neural network architectures.

3.2.1 Neural Network Architectures

3.2.1.1 Simple feed-forward neural network

In a Simple feed-forward neural network, the data flows in one direction from input layers to output layers (with one or more hidden layer between the input and the output layer). Usually, any artificial neural network learns its latent network parameters using back-propagation techniques like gradient descent, and it happens in the reverse direction, from the output layer to the input layer based on the error loss observed between the predicted value and the ground truth. Neurons in the layer 'n' do not have backward connections to any of the previous layers except connections from layer 'n- 1'. Also, they do not have any short-circuit connections to any layers further ahead expect the next layer 'n+1'. This architecture is one of the simplest yet powerful architecture which can learn non-linearity in the input-output mapping effectively. Figure 4 is an example of a simple feed-forward ANN.

3.2.1.2 Recurrent architectures - LSTMs and GRUs

The recurrent neural networks are special architectural variants of artificial neural networks that have been particularly successful with sequential data like speech and language. Time-series data also has sequential patterns hidden in it. Hence, intuitively recurrent architectures are applied to the stock market prediction problem. Long short-term memory (LSTM) [13] networks or Gated recurrent units (GRU) [14] provide some memory element to the simple recurrent architecture which helps network preserve long term dependencies in the sequential data. The simple architecture of RNNs is shown in Figure 5. the x_t in the Figure 5 is input x at time t. The output at time-step t not only depends on input at time-step t but also on the previous inputs until time-step t-1 due to the recurrent design of the network. In simple RNNs, this backward influence dependency is very short, and only inputs that are close to the current input in the sequence affect the current output. To preserve long term dependency of the sequential input, LSTMs and GRU are used which is the case in most of the real world problems involving sequential data. LSTMs and GRUs architectures use memory cell mechanism shown in Figure 6 for LSTM and Figure 7 for GRUs. GRUs are a slightly less complicated variant of LSTMs with a fewer number of gates in the memory cell.

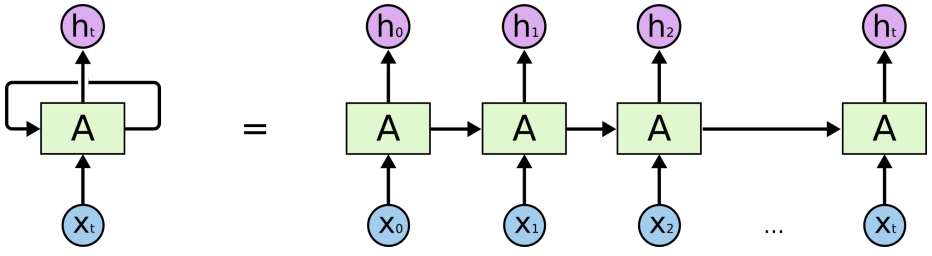


Figure 5: Recurrent Neural Network Architecture.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

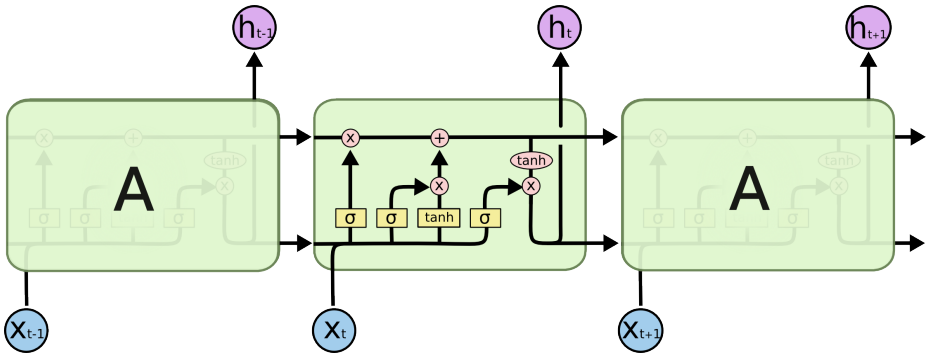
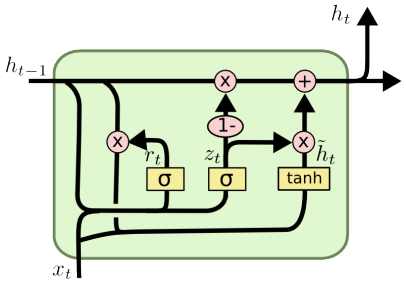


Figure 6: LSTM memory Cell.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 7: GRU memory cell.

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3.2.2 Activation Functions

Each node in the ANN has an activation function. This activation function defines the output of the node. Nodes in the specific layer receive input from the nodes in the previous layers. The output of the node is then generated based on the received input from previous layers. This generated output is then fed to the nodes in the next subsequent layers as per the architectural design. The following activation functions are used in the proposed implementation.

3.2.2.1 Sigmoid/Logistic Activation

This is the most commonly used activation unit for binary classification problem. Sigmoid/logistic activation has range of $(0, 1)$. Value of sigmoid function for input 0 is 0.5. All negative input values have sigmoid value less than 0.5 and all the input values > 0 have sigmoid value greater than 0.5.

$$f(x) = \frac{1}{1 + e^{-x}}$$

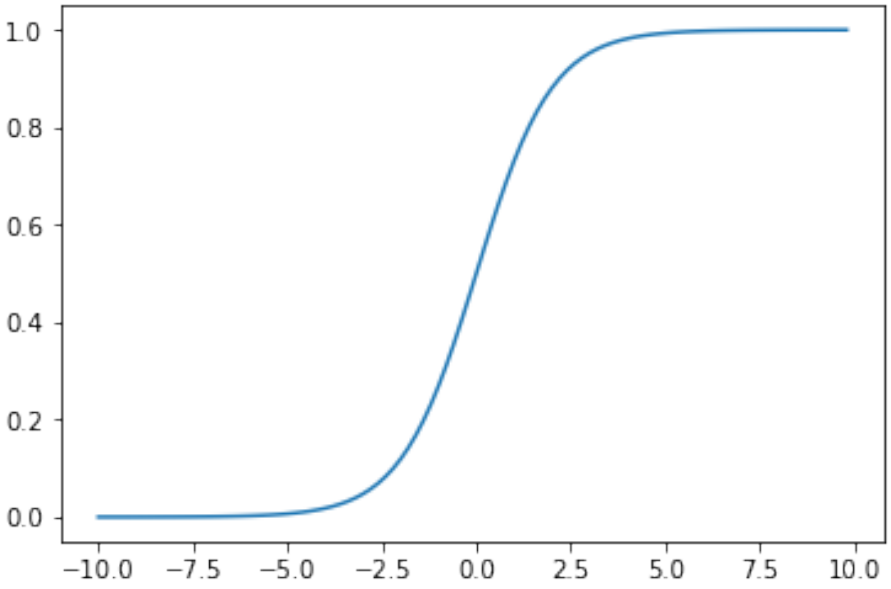


Figure 8: Sigmoid Activation.

3.2.2.2 Hyperbolic Tangent (Tanh) Activation

When the output of the activation unit lies close to 0, the optimization of neural networks becomes reasonably easier. The hyperbolic tangent function is centered about 0, and hence its optimization is comparatively easier than the sigmoid function which is centered about 0.5, i.e. $sigmoid(0) = 0.5$ and $tanh(0) = 0$. the tanh

activation function has range $(-1, +1)$.

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

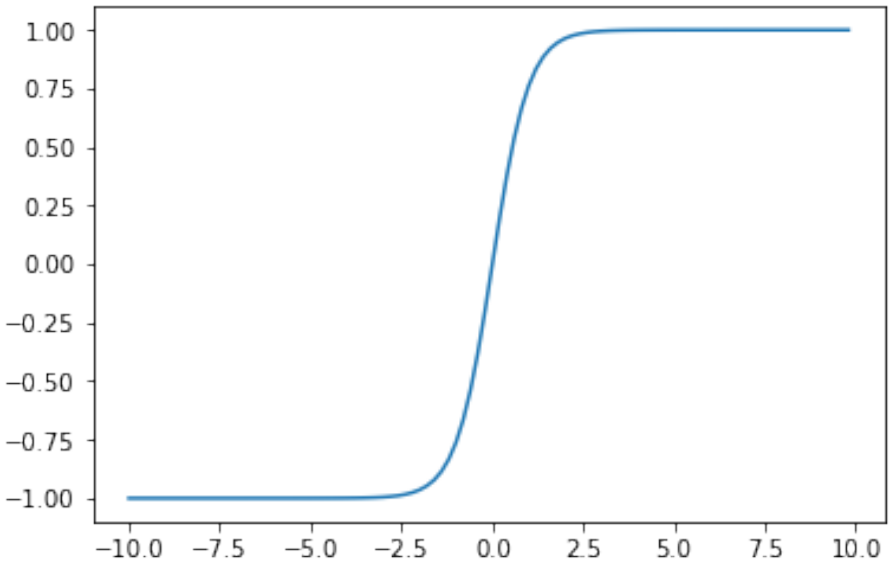


Figure 9: Tanh Activation.

3.2.2.3 Rectilinear unit (ReLu)

Neural networks train and converge much faster if the gradient updates are sufficiently large so that the network can learn something using this gradient. With very high or low input values, sigmoid and tanh functions often wander into the very flat curve area of the activation function with a shallow gradient. This negligible gradient slows down the learning process as the input values either get very large or very small. Hence, rectilinear units (ReLU) functions are often used to tackle this issue. This activation function has range $[0, +\infty)$, i.e., it maps the input values to the range between 0 to $+\infty$ by rounding of negative values to 0.

$$ReLu(X) = max(0, X)$$

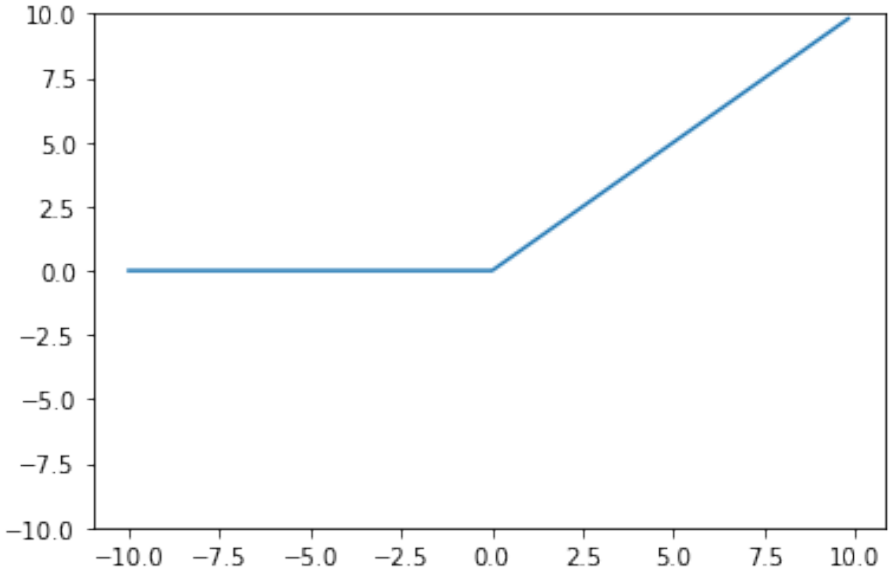


Figure 10: Rectilinear Activation.

3.2.2.4 Softmax

The softmax activation unit is mostly used for multi-class classification problems and is generally used as the activation for the output layer of the neural network.

Softmax function gives probability distribution over the number of target classes. As Softmax function outputs probability distribution, it normalizes all the output values to $[0, 1]$ with all probabilities adding to 1.

The target variable with the highest probability is predicted as a class of the given input. In this project, the target variable with the highest probability will decide the action (buy, sell or hold) agent should take after observing a pattern in current market conditions.

$$\sigma(z)_j = \frac{e^z_j}{\sum_k e^z_k}, \forall j = [1, k]$$

3.2.3 Evaluation Criteria

The overall performance of the agent will be evaluated based on the following two criteria.

- Cumulative Profit: Profit accumulated over all the transactions from the start till current time

- Sharpe Ratio: As per the modern economic theory, the performance of a portfolio cannot be just evaluated based on profits generated. This evaluation also needs to take other alternative investment options into consideration like profits if the initial capital would have been invested in the savings account. Modern portfolio theory suggests that returns need to consider risk while selecting investment options [15].

  The ratio is the average return earned in excess of the risk-free rate per unit of volatility or total risk.

  It measures the performance of the portfolio and adjusts returns as per the excess risk taken by the investor [16].

$$SharpeRatio = \frac{R_p - R_f}{\sigma}$$

where:

R_p = Total return on the portfolio

R_f = Risk free rate (Rate on Savings account or treasury bills)

σ = Total risk or volatility of the portfolio excess returns

CHAPTER 4

Related Works

Despite the argument of the unpredictability of stock market time-series data [3], Most of the economists agree that certain patterns in the financial time series data can be exploited to predict future trends. Data preprocessing is a crucial part of any stock prediction system. High movement, noise, and jumps make the financial time series data highly non-linear and hence to make any predictions, data preprocessing is essential before any statistical or deep learning prediction algorithm is applied. Box and Jenkins [17] devised a method called Auto-regressive integrated moving average (ARIMA) to predict the future average prices using historical stock price series. This method works only on stationary time series data in which the mean and variance of the series do not change drastically over certain repeating periods. Stochastic non-stationary time series needs to be converted to a stationary series with the help of multiple order differentiation to apply this technique. Along with moving averages, other stochastic technical indicators are also used to mitigate the noise and uncertainty in the financial features [18]. These methods have enjoyed great popularity over the last few decades, but they work based on various assumptions and have human defined features which might not accurately represent the current market condition are susceptible to the critical problem of mean-reversion [19] in Figure 11.

Investors can make a profit in intra-day trading due to significant jumps and dips in the time series data (Buy low and sell high). As jumps and drops nullify each other constituting a stable average, ARIMA forecasting can provide a reasonable estimate of the average value and trend of the series, but it cannot take advantage of the volatility of the stock market to make intra-day profits.

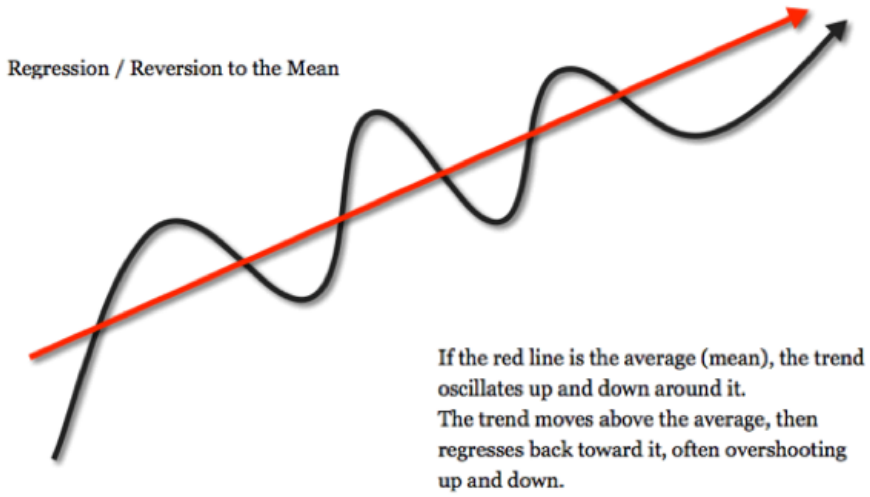


Figure 11: Mean reversion.
Source: http://kattanferrettifinancial.com/mean-reversion/

Most of the stock prediction algorithms developed using machine learning (ML) algorithms to date used discrete historical data as against data received every minute due to the scale of the data. Usmani et al. [20] and Sharma et al. [21] use a wide variety of ML algorithms ranging from simple linear regression to multi-layer perceptron (MLP) to predict the price of stocks in the portfolio. Usmani et.al. [20] use wide variety of external parameters like technical indicators related to market history, news articles related to the stocks in portfolio, sentiment analysis of particular stocks using twitter data, critical commodity prices like oil, Central bank interest rates and strength of the United States Dollar to augment the financial time series data (historical stock prices).

After the breakthroughs in the field of NLP and Computer vision, artificial neural networks suddenly came under the spotlight. The research related to time series analysis has increased many-fold times in recent years. Gao et al. [22] used deep belief networks and principal component analysis on stock technical indicators to forecast

the closing price of the stock on the next day. Vergas et al. [23] experimented with LSTM on stock trend indicators and financial news data to predict future trends. Their research and experimentation concluded that stock trend indicators do not improve the prediction accuracy by much, but the financial news analysis improved the efficiency of prediction by a considerable margin.

All of the above methods used discrete historical time series data points for prediction. But the first major online trading prediction using reinforcement learning on real-time data is proposed by Moody and Saffell [9]. They claim that stock market prediction is a stochastic control problem and it is advantageous to discover strategies directly using direct reinforcement learning without using any value function which forecasts the value of the state in the future. Their proposed method differs from traditional reinforcement learning algorithms like Q-learning and TD-learning in which it attempts to find a value function for this stochastic control problem directly from the current state [9]. They use shallow RNN to keep the state of the previous decisions taken by the prediction system to avoid making excessive transactions. Excessive transactions may lead to extra transaction fees, taxes and slippage costs which might affect the generated profit extensively. Deng et al. [10] extended this highly effective direct reinforcement learning approach to a Deep Direct reinforcement learning approach by adding a Deep neural network for dynamic feature extraction before the data is fed to the direct reinforcement learning module (a shallow recurrent neural network). Deng et al. [10] also used fuzzy extensions to reduce the uncertainties in this time series data. These fuzzy representations are then passed to Auto-Encoders for compressed and compact feature representation to be fed to the RL module.

Other stock prediction models using reinforcement learning include efforts from D. Lu [24]. Lu implemented his agent using LSTMs to generate embeddings instead

of Auto-encoders used by Deng et al. [10]. Dropouts are added to the deep RNNs to avoid the issue of vanishing gradient in the research conducted by Lu [24].Both Deng et.al. [10] and Lu [24] use real-time stock market data.

Out of curiosity, this project will explore if we can build an intelligent stock trading agent who will be able to make decisions of buying and selling the stock based on some policy learned during the training. Our goal will be to incremental development of this agent who will handle a single security at first followed by the ability to manage a portfolio of stocks. Most of the approaches mentioned in this chapter work only with individual security/stock. Once our agent is able to perform well on a single stock, the subsequent goal would be to analyze possible training methods for stock prediction agent for multiple stock portfolio using various reinforcement learning algorithms, which scale better with a minimal trade-off in the accuracy. Moreover, this project aims to create a complex, yet flexible, system such that specific components of the algorithm can be modified without changing other parts of the system.

CHAPTER 5

Data-sets, Features, and Preprocessing

Financial time series data related to the stock prices are available in abundance, and the scale of this data is huge. This data can be categorized into two categories:

- Discrete Stock Market Data: (Historical stock price values)
  - Features
    - Open: the Opening price of the stock on a particular date
    - Low: Lowest price of the stock on a particular date
    - High: the Highest price of the stock on a particular date
    - Close: the Closing price of the stock on a particular date
  - Yahoo Finance Dataset [25]
    - This dataset contains historical prices (With attributes mentioned above), dividends and split data for most of the large 500 American companies on Standard & Poor's 500 index. Historical prices dating back to 1970 can be obtained using Yahoo Finance Data or API.
    - Data can be downloaded in CSV or can be acquired in JSON format using API. Data preprocessing on both JSON and CSV formats are quite convenient as multiple programming languages nowadays are packaged with in-build libraries for various operations for such file/-data formats.
  - Quandl [26]

    * Quandl is a datastore with more than 1 million datasets which are free and open to all. It contains trading data gathered from multiple credible sources. Quandl has a free universal access API (with rate limiter per second), which can be used to fetch any stock price time-series.

- Real-time Stock Market Data
  - Real time stock prices and technical indicators can be obtained using Alpha Vantage API [27].
  - Technical indicators mentioned in Chapter 3 can be used to augment the trader bot to take decisions based on even more refined data in addition to the time series data.
  - Quandl [26] API is one alternative for Alpha Vantage real-time data API which can also be used to fetch the financial information real-time.

5.1 Initial Data Analysis

Initially, this project was aimed at analyzing real-time stock trading data and training the model on the same. But due to the extremely high volume of real-time data (approximately 65000 ticks for a single day) and limited computational and data storage resources on the local machine and free Google Cloud Compute instance, a fallback strategy of using discrete historical stock price time-series data was used for the proposed experimentation.

5.2 Data Visualization

It is imperative to visualize the data before applying any machine learning techniques to the data. Data visualization provides important data insights that might

help users select proper algorithms and hyper-parameters to improve the performance of the prediction system or the agent. For stock prices, simple graphs provide good insight into price movement and trend. Experiments in this project are conducted on closing prices of stock in the various date-time ranges. Figure 12 is a simple visualization for the sample stock market time-series data.

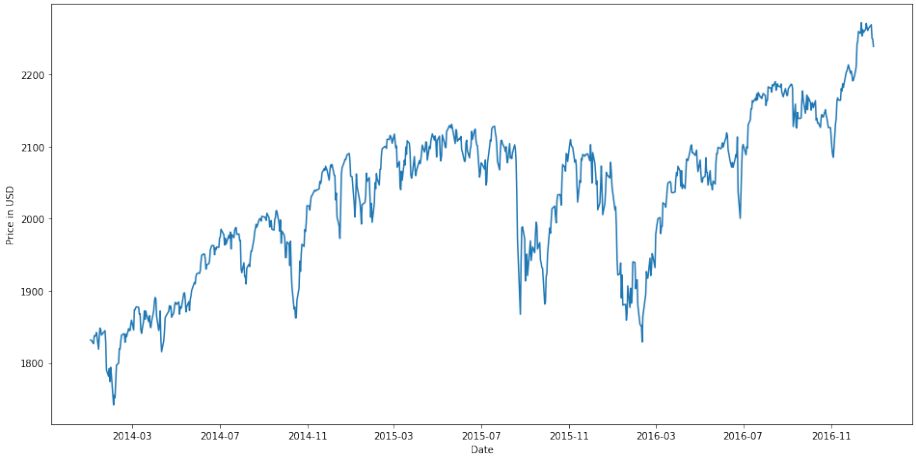


Figure 12: S&P 500 Index from 2014-2016.

5.3 Data pre-processing

The quality of the data highly influences the performance of the machine learning algorithms. Various data-preprocessing techniques like data cleaning, data normalization are essential and usually the first part of the pipeline of any machine learning project. For this project, the following two preprocessing methods are used.

5.3.1 Scaling

Scaling is a type of Normalization which is a ubiquitous data preprocessing technique used to scale the data values between a specific range. Cleaner and normalized data help deep learning algorithms converge much better and faster compared to the raw stock values. The following figure shows a scaled conversion of original raw values

in the above graph to a range of 0-1. Scaling values close to zero helps neural network optimize comparatively faster.

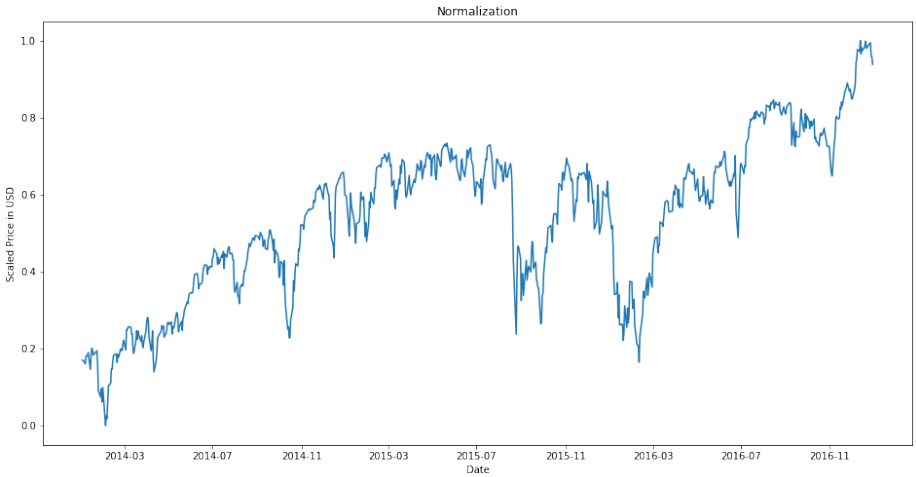


Figure 13: Scaled S&P 500 Prices for 2014-2016.

5.3.2 De-trending

In time series models, data sometimes follows a linear trend that might infuse some bias in the model while learning the strategy. Multiple strategies can be used to remove the linear trend. The Figure 14 and Figure 15 show how the raw data and scaled data look after the removal of the linear trend respectively. This technique of trend removal helps the agent take proper policy decisions regarding the optimal action without getting affected by the specific pattern. Removal of trend helps the agent to learn a policy about the volatile movement of stock even in the trending market.

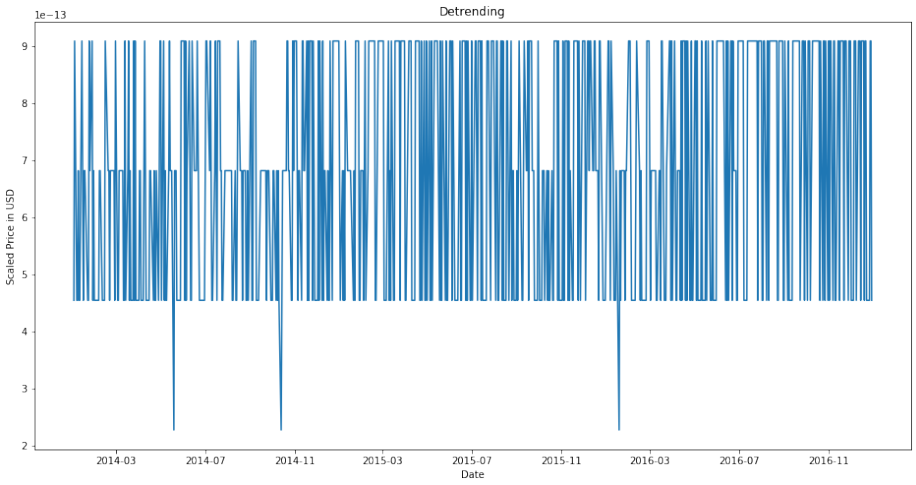


Figure 14: S&P 500 Index from 2014-2016 after removing trend from raw time-series values.

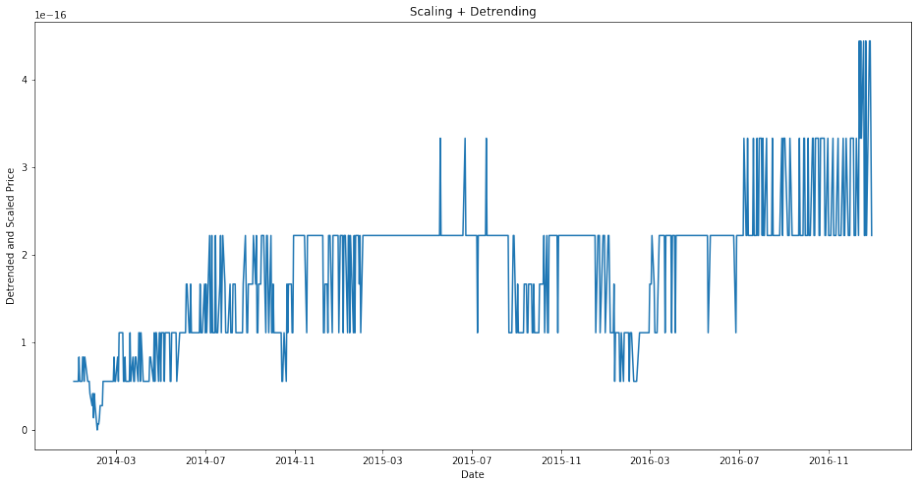


Figure 15: S&P 500 Index from 2014-2016 after removing trend of scaled values.

CHAPTER 6

Experiments and Analysis

6.1 Trend Prediction

The recurrent neural networks and more precisely Long-short term memory (LSTM) networks are widely used for time-series data, and hence LSTMs were tested on the stock market data to get some insights and even predict the future prices based on a time-window of the last few days.

The graph in Figure 16 shows the predicted (orange) vs. real value (blue) of the scaled S&P500 index for LSTM. As the stock value space is enormous, predicting the exact value of the future stock price is nearly impossible. Additionally, multiple factors, other than past stock values, influence the value of the stock on the next day. Hence, most of the deep neural network stock prediction systems try to predict the trend of the stock movement based on past stock values and also some other indicators, including news articles and technical indicators. These predicted next day values help investors make an informed decision regarding the action to take for that particular stock.

Even though LSTMs try to map non-linearity in the time-series data much efficiently, they face few problems. As seen in Figure 17 LSTMs may suffer some lag in the time series forecasting and in high entropy environments like the stock market, an investor may suffer huge losses due to such lag. Furthermore, a simple trend prediction system does not suggest which action is the appropriate action to take at the proper moment. Additional logic needs to be applied to the predicted next day value series to generate decision policy. So the question is, can we do better and get our

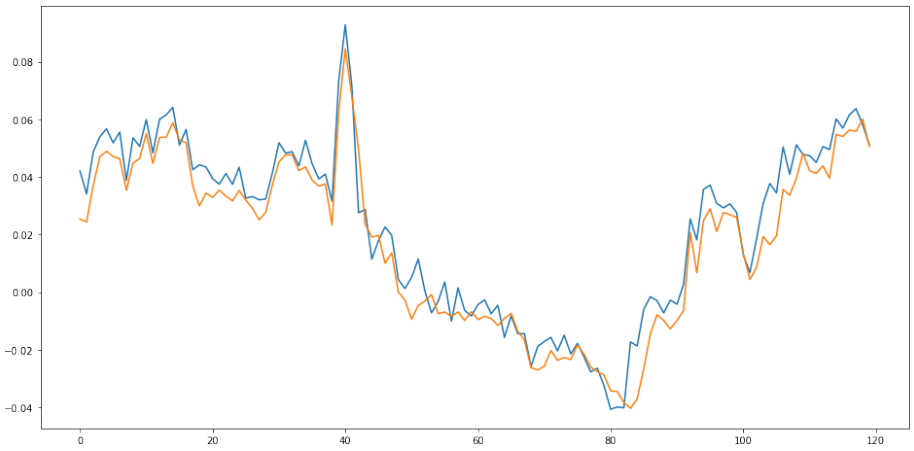


Figure 16: S&P 500 Index Stock price trend prediction using LSTM.

model to learn the policy instead of just a trend?

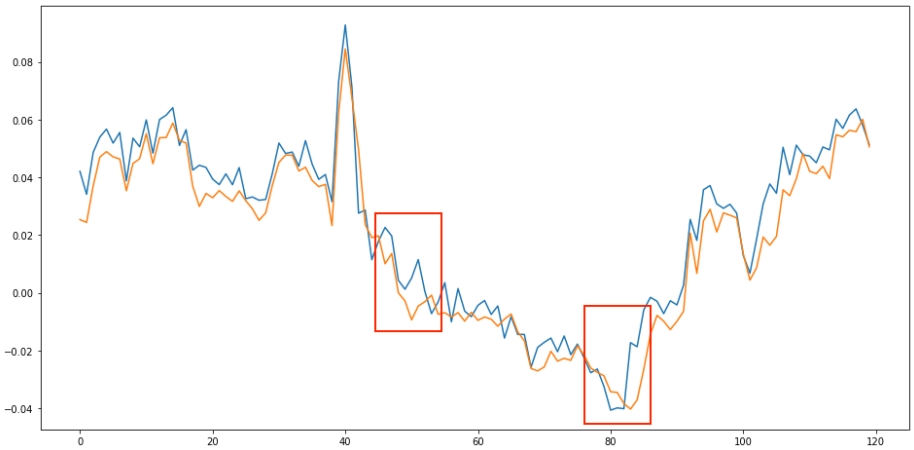


Figure 17: Stock Trend Prediction delay in LSTM.

The next few sections shed more light on the trading policy learning experiments.

The effectiveness of any reinforcement learning algorithm heavily depends upon how the problem is formulated. It is hard to train an agent for games with complex rules and generalized objectives. If the problem setting has constrained objectives, and thus a smaller search space, the agent will be able to learn the decision policy quickly and will have considerable accuracy. Moreover, following such a policy, the

agent might yield reasonable results for the selected problem. Generated decision policy might not be the optimal policy, but it helps the agent learn basic rules of the game, and later the agent can be retrained using this basic decision policy to learn some complex rules.

All subsequent decision policy experiments will follow a similar structure where initially a problem setting will be defined. Results with the specified setting will follow the constraints defined in the problem setting section.

6.2 Random Decision Making Policy

Initially, we try a simple random decision making policy to demonstrate later that we can do learn comparatively better policy using RL. This project aims to find an optimal decision policy for stock trading and other commodity training. As per the random walk hypothesis [3], this simple approach would be to take the actions randomly like a blindfolded monkey and to observe the profits generated over the selected testing period.

6.2.1 Problem setting

For random decision policy, the constraints are the following:

- We keep the problem definition simple, with the agent dealing with the stock of just one company and starting with virtually unlimited base capital.
- The buying strategy is that the agent will buy a single stock when 'Buy' action is sampled randomly from the action space.
- At the time of selling, all shares of that stock in the inventory/portfolio will be sold.

- Buying and holding will have virtually no rewards.
- Selling reward is the difference between the possible valuation of the complete portfolio using current stock price at the current time-step and the actual total value of the current portfolio at the current time-step.

Rewards will be accumulated over all time-steps in the given training/test window for a single episode and will be reset at the start of the new episode. (The random decision policy will not have any training data. Training data is equivalent to testing data if we are not going to train any policy on it.)

6.2.2 Results

Figure. 18 shows profits vs episode graph for this random decision policy.

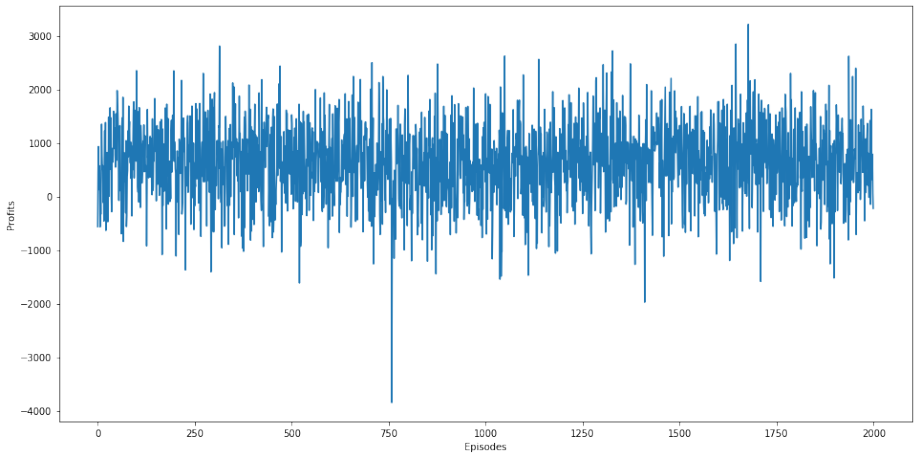


Figure 18: Random Decision Policy Profits distribution after 2000 Episodes

The agent has no control at all over the profit maximization. The random actions lead to scattered profit and loss distribution where our agent accumulated loss in many simulations. From Figure 18 one can see that the profit distribution over episodes is random and may result in a loss if the agent follows this policy.

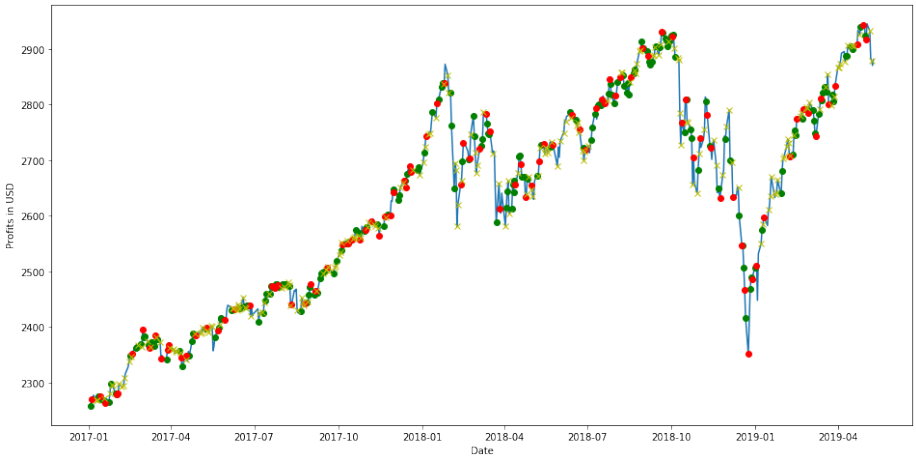


Figure 19: Random Policy Action on one episode.

Actions taken according to the random policy in a single randomly selected episode are presented in Figure 19, where green dots represent action 'Buy', red dots represent 'Sell' and yellow 'x's represent 'Hold' Position. The agent is not following the winning rule of 'buy low and sell high' in this stock market environment. This can be observed in Fig. 19 where the agent is sometimes selling stocks in the deep troughs which can be considered an excellent buying opportunity instead. The episode in the Figure 19 achieved a profit of 663.73974609375.

This policy does not seem to maximize the profits for every episode as it is randomly selecting buy, sell or hold actions without considering the state of the environment and without following a specific strategy or in this case optimal policy.

6.3 Deep Q-learning

We want our agent to learn some optimal policy, which should be able to learn the rules of trading irrespective of the type of the stock data fed to the system or portfolio of several stocks that is provided to the agent. As described in Chapter 3.1, the deep Q-learning algorithm is used to train an agent to learn the optimal policy.

The following subsections describe problem setting and experiments with this policy.

6.3.1 Problem Setting

The problem setting is extremely crucial in a deep Q-learning based on reinforcement learning and will often determine the success (mostly in terms of convergence and accuracy) of the trained policy model. As the process of training the agent with a large continuous action space is challenging, we imposed the following constraints.

- The stock data for only one company was selected for training the agent. In real life, hedge funds and big investment firms diversify their portfolios to minimize the risk associated with a single company stock.
- Buy and hold rewards were considered zero initially, but later some modifications were made to these rewards, after some failed experiments.
- The reward for 'sell' action was determined using stock buy and sell settings mentioned in Table 1.

We experimented with the following settings for the buy-sell mechanism, and the 'sell' action reward was computed according to the selected configuration.

Table 1: Different Buy-sell mechanism

Buy Scheme	Sell-Scheme
Buy 1 stock unit	Sell 1 most recently bought stock unit in the portfolio
Buy 1 stock unit	Sell 1 stock unit in the portfolio with lowest buy value
Buy 1 stock unit	Sell 1 stock unit in the portfolio with highest buy value
Buy 1 stock unit	Sell all stock units in the portfolio

Reward computation for 'sell' action can be intuitively deduced from the buy-sell mechanism in Table 1.

Training our agent to buy a variable number of stocks and selling a variable number of stocks at a particular time-step will lead to a tough deep q-learning problem with huge action space and will take a longer training period with no guarantees of convergence (with non-linear function approximator [28], [29]) or optimal policy. Hence, the simplified settings mentioned in Table 1 was used to train the agent.

A four-layered feed-forward neural network is used as the value function approximator for the Deep Q-network. The first layer is the input layer with the last 30 day's closing price for selected stock fed as an input. The next three hidden layers have 128, 128 and 64 neurons respectively. The final hidden layer outputs 3 Q-values for three associated actions for the given state. Rectilinear activation units were used for the hidden layers to solve the problem of vanishing gradient descent while training the network. The output layer with three neurons has a linear activation unit for Q-value outputs. The action with the highest q-value is selected for the specific state in the Q-learning algorithm in Chapter 3.1.

6.3.2 Convergence of Q-network with Deep Q-learning

When non-linear function approximators are used along with model-free learning algorithms, like deep q-learning, q-networks (the neural networks for q-values), to predict value functions they may diverge causing difficulty to learn a stable policy [28] [29]. Mostly linear function approximators are used for convergence purposes but learning complex non-linear functions in the complex stock environment using linear function approximators is hard.

6.3.3 Experiments

Initially, the agent was trained on S&P 500 index values between the years 2006 and 2010. The simple idea behind this training window was, if the agent can learn to profit of the time window with high volatility and uncertainty(financial crisis of 2008 and its aftermath), it will surely do well in stable periods. But this hypothesis failed and the agent learned a strategy which was extremely defensive and it failed to converge on a stable policy. The agent suggested to sell everything and exit the market as shown in Figure 20. Red dots in Figure 20 are the time-steps where agent selected the 'sell' action. It frequented the most cautious strategy learned in the training run.

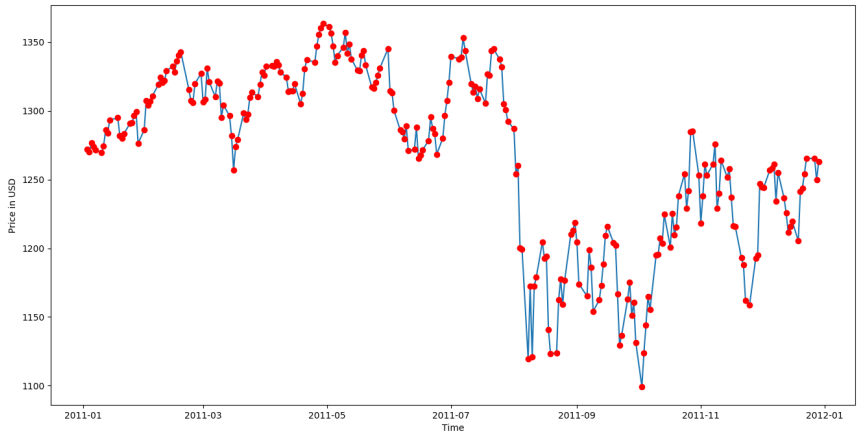


Figure 20: Agent selling everything from start: Q-network convergence failure.

Further, it was also observed that just by increasing the training time window, did not help the agent learn much either. It again failed to converge and continued to follow a specific pattern of action, without maximizing the reward properly and learning a robust policy. The agent went on a buying spree with occasional hold positions as shown in Figure 21 with green dots showing time-steps where agent

selected 'Buy' action and yellow 'X's represent 'Hold' action, when the trained model was run on the same test period for S&P500 index values for the year 2011. This was observed because the base capital assigned to the agent had no constraints over it.

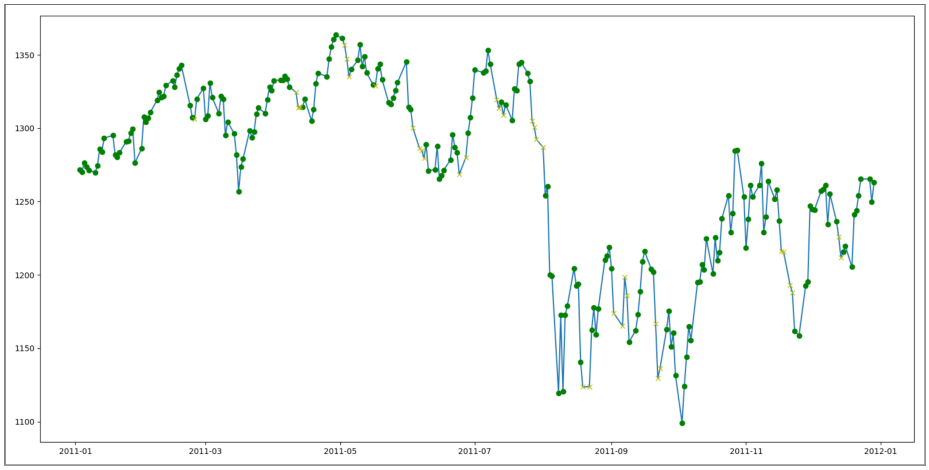


Figure 21: Agent selling everything from start: Q-network convergence failure.

To help the deep q-learning agent learn better and converge faster, experience replay mechanism and Fixed Q-targets [8] were incorporated into this system. Experience replay mechanism used the mini-batch of size 64 randomly sampled from seen $\langle$ state, action, reward, next state$\rangle$ samples. Q-Target neural network was kept fixed for ten episodes to let Q-estimation reach as close to the Q-target value as possible. After ten episodes, weights of the Q-estimation neural network were copied to the Q-target policy neural network.

The agent was then again trained on S&P500 index data from 2005 to 2017. This time range was considered long enough with a variety of possible stock market time-series transitions. This time-window had an initial period of upwards trend from 2005-2008, a volatile period of 2008-2011 (financial crisis of 2008) followed by a steady bullish market of 2012-2014 and finally an uncertain and unstable period of 2014-2017. The test period for the trained model was selected to be the data for the

year 2011 and the duration between 2018 - April 2019.

Unfortunately, the generated model with the improved deep Q-networks again faced some converge issues, where the model sometimes used some of the policies more frequently in the initial runs and settled on those policies for any time-step.

Designing a reward function for a volatile high entropy stock market environment was a tricky part. To discourage the agent from taking the same action for long stretches of periods and force it to participate in the market, large negative rewards were assigned to the successive buys, serial holds, and continuous empty sells. Continuing empty sells are when the agent is trying to sell the stock unit when it does not have any units in its portfolio.

Initially, a fixed base capital constraint was considered in the problem setting for the agent to avoid a continuous buying spree, but the agent started bankrupting itself in multiple episodes and started deviating towards 'sell all and exit the market' strategy. To make the training simpler, finally, unlimited capital was assigned to the agent. Even though the base capital was unlimited, consecutive action penalties forced the agent to change positions with proper intervals to generate profits and not exhaust and abuse the unlimited base capital.

The commission factor was also ignored in this deep q-learning setting to keep the problem more straightforward. Due to this, the agent frequently changed positions to make a profit with almost next to zero hold positions.

The commission factor is used in policy gradients reinforcement learning in the next section.

After making numerous adjustments to the value function approximator and reward assignment, a reasonable model was trained. This trained model avoided

losses as shown in Figure 22, but still started diverging again after a certain number of episodes. The episode number, where the model performed better, was obtained heuristically using a trial-and-error approach.

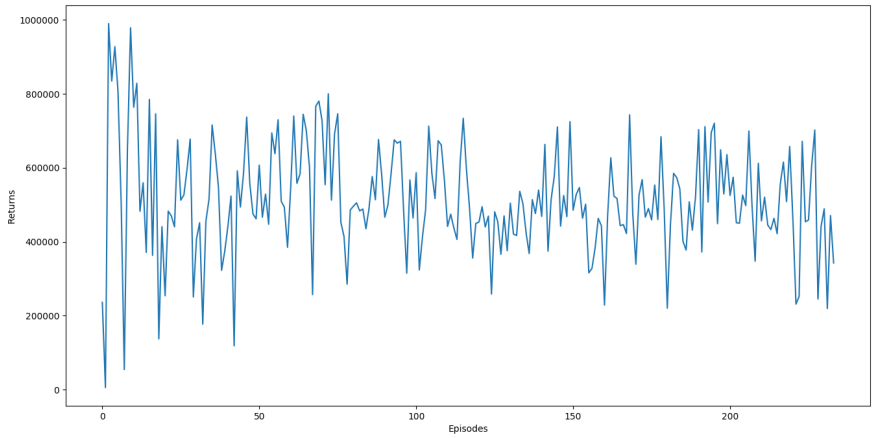


Figure 22: Profit distribution using Deep Q-networks over 250 episodes

This episode number threshold was then again tested using the early-stopping mechanism to verify the performance of the learned policy.

The following graph is generated using model after episode 100. After 100 episodes, the model again started diverging and started taking one action for all days over the other 2 actions in the action space.

First, this model was tested on the same stock, S&P500 index value for 2 test periods, for stock prices in the year 2011, Figure 23 and then for prices in the period January 2018 to April 2019, Fig. 24.

Both of the plots in Figure 23 and Figure 24 seem to justify the reasonable strategy learned by the agent, where it is trying to buy the stock when it senses the troughs, and it is trying to sell the stock when it detects peak values for S&P 500 index stock price.

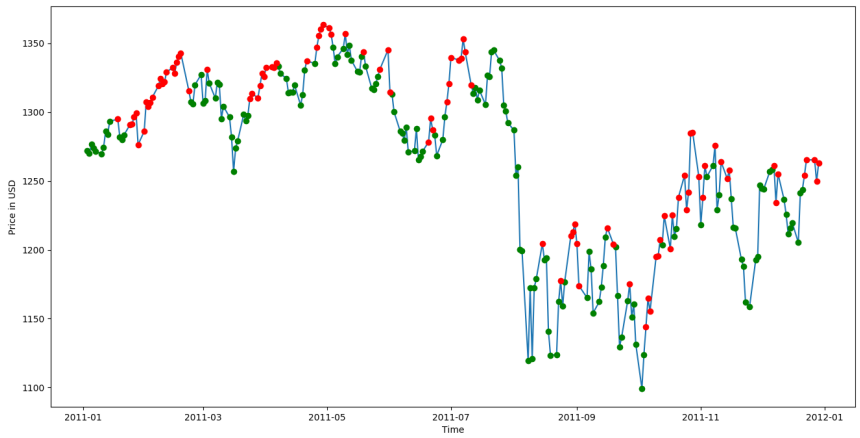


Figure 23: S&P500 Index Action plot using DQN decision policy: 2011.

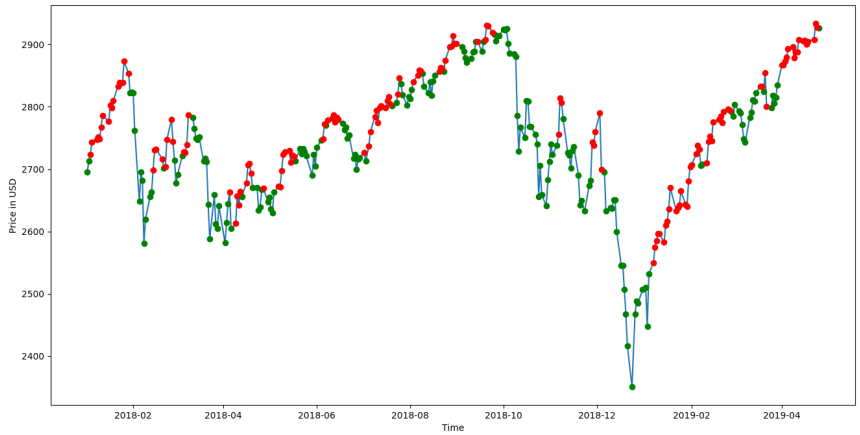


Figure 24: S&P500 Index Action plot using DQN decision policy: 2018 - Present.

Numerical profit values of all the experiments are documented in Table 2.

6.3.4 Other interesting experiments

We hypothesized that if the agent learns the rules of the game, it should be able to play the games with other stock data also. The trained policy was tested on other stocks like Alphabet(GOOG), the parent company of Google. Fig. 25 shows

the performance of policy for GOOG stock prices in the year 2011 and Fig. 26 shows learned policy performance for GOOG stock prices in range January 2018 - April 2019.

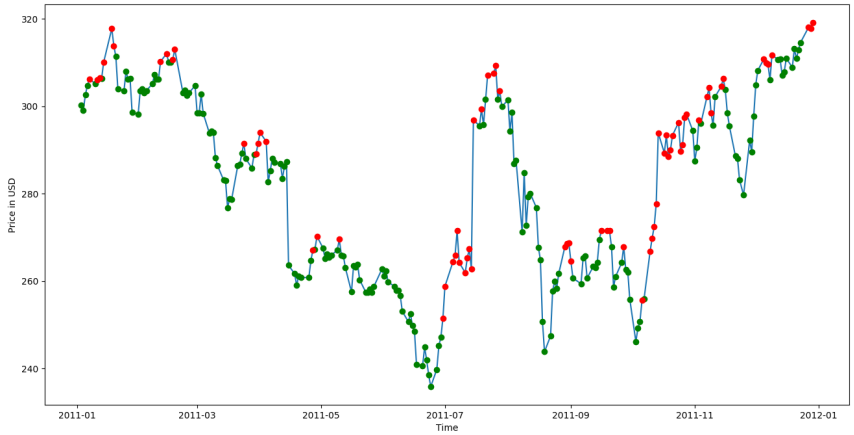


Figure 25: Google stock price Action plot using DQN decision policy: 2011.

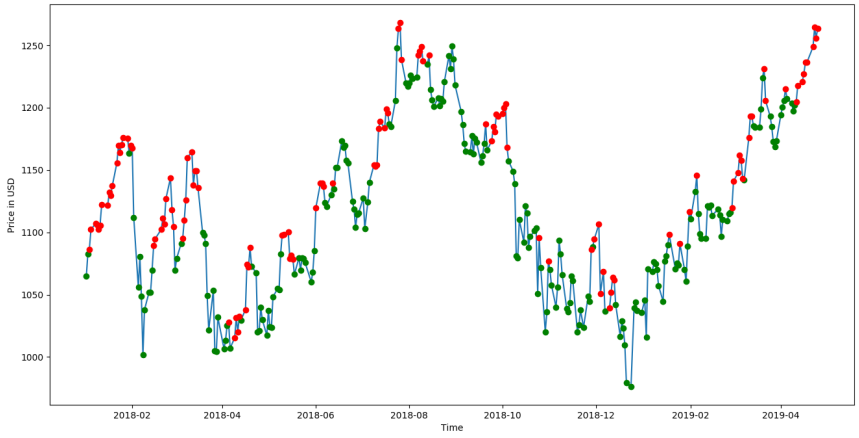


Figure 26: Google Stock price Action plot using DQN decision policy: 2018 - Present.

The DQN decision policy learned on S&P 500 index values generalizes fairly well for Google stock price, and it is still playing the game of 'Buy Low and Sell High'

pretty well by catching troughs and peaks with decent accuracy as shown in Fig. 25 and Fig. 26.

One possible justification for this generalization and a nice model fit for Google stock price might be because, with high market capitalization, the movement of Google stock influences the S&P index in comparatively higher proportion than other stocks. And hence to see the efficiency of the learned model we tested this model for Gold commodity prices.

Gold prices are usually considered inversely proportional to the stock market indices. This possible co-relation is not proven, but empirically investors tend to pull the money out of the stock markets to invest it in other lucrative investments like gold. we tested the learned DQN decision policy model on gold prices for the same test period as we used for S&P500 index and Google stock and the results are presented in Fig. 27 and Fig. 28

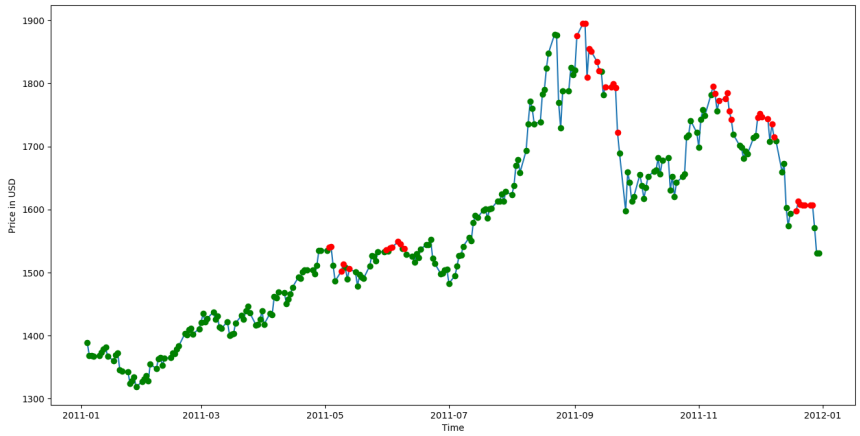


Figure 27: Gold Price decisions using DQN decision policy: 2011.

In the year 2011 when stock markets were still reeling from the effects of the aftermath of the financial crisis of 2008, agent stockpiled gold and still tried to sell

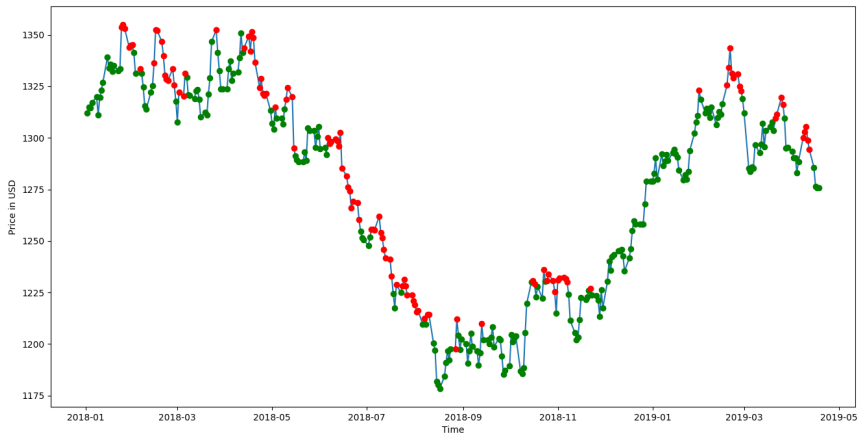


Figure 28: Gold Price decisions using DQN decision policy: 2018 - Present.

off the gold when it sensed peak in the Gold price time series in Fig. 27.

For recent years (2018-19) gold prices chart, the agent was not that efficient but still not extremely bad either. It had difficulty sensing the bear gold market during periods of May 2018 - Sept. 2018. But once the prices hit bottom, the agent started buying and tried to profit off the purchased gold during November 2018 as shown in Fig. 28.

Table 2: Profits for stocks over selected time period

Stock	Time Period	Portfolio value	Transaction Profit	Overall Profit
S&P500 Index	2010-2011	81371.21	-2308.12	1837.47
S&P500 Index	2018-2019	120245.98	2653.2	14701.81
GOOG	2010-2011	29757.29	-40.68	5176.109
GOOG	2018-2019	123262.72	3102.97	22324.42
Gold (WGC)	2008-2011	711741.85	-75295.25	-219253.35
Gold (WGC)	2010-2011	265350.0	1233.00	-18154.5
Gold (WGC)	2018-2019	29757.29	-40.68	4295.9

As supported by the last few experiments, designing a reward function for the

stock market data is extremely tricky and challenging. Further, with Deep Q-learning we learn deterministic policies, i.e., given a state representation, the Q-network selects the action which gives maximum Q-value for a specific state-action pair. This greedy policy for action selection also contributes to convergence issues as there is a high possibility that the agent might oscillate between two states indefinitely causing training issues.

6.4 Policy Gradients and Direct Reinforcement Learning

After facing so many issues and so numerous parameter experimentation with Deep Q-learning, we tried testing the agent training with policy gradient methods. The main reason behind trying these methods was to add some real-world constraints to the learning process and decision policy of the agent. Policy gradient methods intuitively fit the problem of stock trading more better because these methods do not rely on future expected reward computation using complicated Bellman equations and hence, calculation of complex and tricky value function. Experimentation ideas were borrowed from different policy gradient algorithm variants [30], [10] and [24]. As mentioned in chapter 3, we directly parameterized the policy and update the policy using gradient methods. The following subsection describes the problem settings and experimental results with these methods.

6.4.1 Problem Setting

The main disadvantage with Q-learning is that the Q-network tries to provide the deterministic action with the highest q-value for the current state representation and we need to calculate the q-value using value function which needs computation of the future expected return from the next state. This expected future return prediction

in high-entropy environments is challenging and quite often unpredictable. In policy gradient methods, we will compute reward directly based on the current action taken and the reward achieved at the current state. As we are training this agent on historical data, we can play this game as a series of episodes and perform episodic gradient update to our behavior policy. Policy gradient methods and it's variant techniques try to predict probability distribution over action space. The agent then decides which action to take based on the probability distribution (stochastic policy) in a particular state.

With the primary objective in mind, we first decided to see if we can scale the agent to handle two different stocks in our portfolio. The agent had 5 actions, hold, buy first stock, sell the first stock, buy other stock, sell other stock. A few other experiments with six actions with the action 'Hold' split into 'hold first stock' and 'hold second stock' were also in-progress at the time of submission of this draft. But this report will only include results with two stocks and the corresponding five actions mentioned above.

In this setting, we have used base capital and commission constraints while training the agent. The buy-sell mechanism used in this scheme is also similar to the one used in deep q-network implementation. Our agent will buy one stock and sell one stock from the inventory with the minimum value when corresponding action is sampled from the stochastic probability distribution for the action space.

The neural network used for the agent in policy gradient methods (Slight mix of simple Actor based method and direct reinforcement learning agent) will be as follows. Initial Input layer and its size will depend on the number of inputs we feed to the system (historical data and other additional features). If we use historical data, then the next layer will be either gated recurrent unit (GRU) or Long short-

term memory (LSTM) unit. If required we can stack the deep LSTMs with either 2 or 3 layers but this will increase the training time and will, in turn, lead to increased complexity. If we have any additional features along with historical prices like any technical indicators, e.g., moving averages or momentum indicator, we will replace LSTMs with a fully connected layer. This first hidden layer will have either sigmoid or tanh activation, and it will be followed by two more fully connected layers of 128 and 64 neurons respectively with ReLu activation. This 64 neuron layer will be connected to two output layers, one of which will be connected to a softmax layer with the number of neurons equal to the action space and the other one will be a linear layer with one neuron outputting the value for the state.

The following are the experiments and interesting observations for this setup.

6.4.2 Experiments

Stocks in the portfolio were selected based on their correlation. The test period for selected stocks was from 2014-2019. The first experiment was between the Apple(AAPL) and the Microsoft(MSFT) stocks.

In the above experiment, the agent was able to achieve a profit of 3313.4180 when the base capital provided was 15000. The inherent problem with the system is that the agent can make only one decision per time-step and if the stocks are reasonably correlated, inversely to be more precise, then agent was forced to choose action related to one stock only. So, if one stock had a peak and at the same time the other one had a trough, then the agent was forced to pick one of the two actions and hence was not able to maximize the overall portfolio profit.

The amount of initial base capital also played a crucial part. With low initial base capital, the agent bankrupted itself in most of the episodes during the training

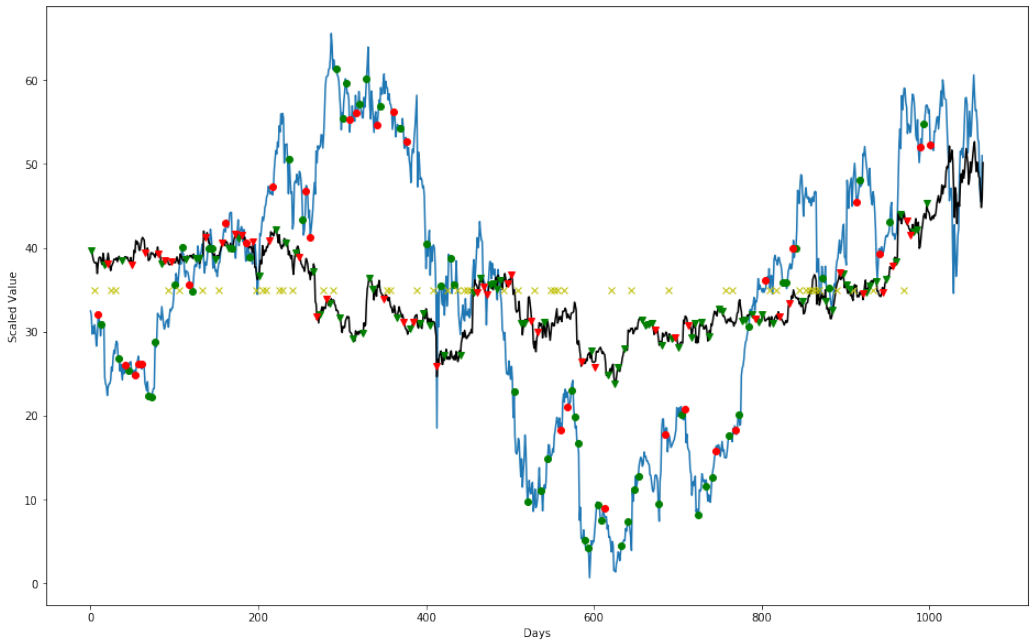


Figure 29: Apple-Microsoft Scaled price Action plot using Policy gradients: 2014 - 2019.

Figure 30: Apple-Microsoft price Action plot using Policy gradients: 2014 - 2019.

period and performed poorly with slight losses on the books during testing time. A significant default commission cost of 10% was assigned to deter the agent from frequently changing positions. This cost was reduced gradually (from 10% to 8% and then to 5%)during the experimentation phase to force agent make more transactions when it had the capital to invest.

This agent at least learned to survive the volatile stock market and simultaneously avoid losses (with profits of 15% to 22%) using this method of reinforcement learning for these particular stocks.

The importance of the base capital hyper-parameter was again observed when the agent was trained on stock prices of two oil companies, Chevron and Exxon. Due to extreme volatility, the agent was not able to survive on low base capital. Even with a considerable base capital, it could not make significant profits. Its profit range hovered around 2% to 5%. Unlike previous stocks, these two stocks had considerably similar stock price movement and comparatively the same stock unit value. Hence it was expected that agent will perform better on this pair of stocks.

This model trained on detrended stock price data for oil companies could not even generalize well on real stock price data before preprocessing.

Simple actor based policy methods usually face the issue of high variance and slow converge. The above models were trained on 5000 episodes. Increasing the number of training episodes to a large number (100000 to 1M) might improve the performance of the agent.

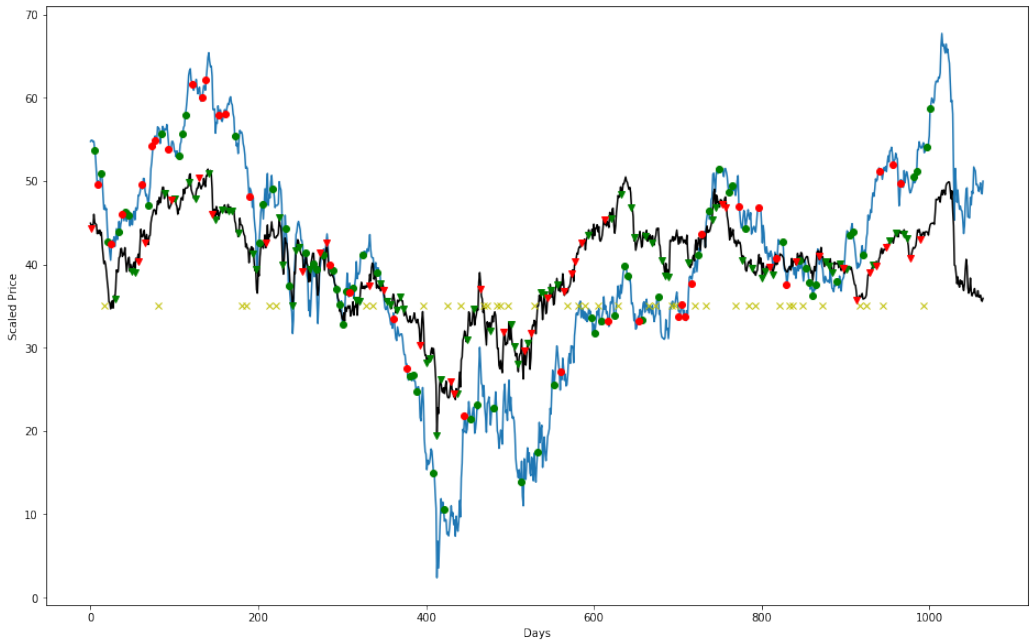


Figure 31: Chevron-Exxon scaled price Action plot using modified Policy gradients: 2014 - 2019.

Figure 32: Chevron-Exxon original price Action plot using modified Policy gradients: 2014 - 2019.

CHAPTER 7

Conclusion

From the number of permutations and combinations with constraint parameters, we observe that it is tough to come up with a generalized reinforcement learning agent with an optimal trading policy which works in any scenario and in a considerably volatile environment.

From the initial experiments, we observed the potential of using reinforcement learning for playing the game of stock markets over conventional and recent statistical, machine learning and deep learning methods. Simple LSTMs proved to suffer from delayed prediction problems where lagged prediction led to substantial mismatches in the price trends. Simple trend analysis with deep learning was inconclusive in determining positions of buy, sell and hold to maximize the profits, and hence in such cases, investors need to devise a strategy based on the predicted future values and trend.

The random decision policy for selecting the position of buy, sell and hold randomly was not that efficient, and even if it generated considerable profits in multiple training runs, it also leads to extreme losses in a nearly equal number of runs. The distribution of profits versus episodes was unstable(19), and the requirement was to train an agent that performs consistently over a specified duration of time on stock time series data.

It is hard to train a generalized agent using a deep Q-network. We were required to properly balance the constraints through trial-and-error methods (survival versus profit maximization and then trying to get both) to let the agent learn optimal policy

to make profits in the stock market with the agent honoring all set of rules decided for this game. Deep Q-networks faced the problem of convergence when we used non-linear function approximators like neural networks. Designing a reward function in high-entropy environments in a tricky and challenging task. To make the agent learn some meaningful strategy, we were required to make numerous tweaks to the reward function based on its performance in the previous runs. Increasing the action space of the agent made training extremely hard for the agent. The agent trained on S&P500 index data with one stock unit buy and one stock unit sell (with the minimum value from the inventory) performed reasonably well on different stock-market time duration price data. To get these results, we relaxed the survival constraint for the agent and allowed it to operate with unlimited base capital.

Further, to keep the problem simple, we observed that relaxing commission fee parameter also helped the agent train faster at the expense of frequent position changes and very few 'Hold' positions throughout the run. This trained agent was able to generalize reasonably well on other stock data and also on presumably inversely proportional commodity, gold. The agent learned the rule of the game reasonably well and picked peaks and trough signals with considerable accuracy.

Later we experimented with policy gradients for training this agent on the same training data. Policy gradient allowed us to add a few more extra training parameters like commission and starting base capital. It intuitively made sense to immediately get the rewards in such time series data rather than tapping into the future expected rewards from s specific state. Simple Actor-Critic method agent was trained for pairs of correlated stocks to see if the agent can learn the rules of the game in the slightly tricky game with larger action space.

Policy gradient methods helped the agent learn the rules of the game better when

we added some additional constraints like base capital and commission penalties. The agent learned to survive the game, but at times it was not able to maximize the profit while doing so. A slight variant of Actor-critic policy gradient and direct reinforcement learning was not able to perform as exceptionally well as it is documented for single stock in [10]. The addition of second stock increases the complexity of the system. We observed that selecting a single action at a specific time-step for any one of the stocks in the portfolio led to a sub-par agent learning. The most appropriate way for training an agent to handle many stocks should be taking multiple actions (one of the buy, sell or hold for each stock) at every time step to optimize the overall stock portfolio performance.

The training time series data should have all possible time-series patterns so that agents can learn by interacting with such an environment. Training the agent on extremely volatile time-series periods did not help it perform well on comparatively stable and trending periods. The selection of proper training period (total period) and training window for each day remains an open question. We trained the model on historical stock prices with discrete time time-steps (Close value for each day). The scale of the real-time data makes training RL agent who acts in an on-line manner a game on a whole new level with massive compute and storage requirements.

And finally, there are multiple other factors like company-related news, internal company documents, company earning reports (to name a few) that also play a crucial part in driving the value of stock in one direction or another. An agent trained in an environment so constrained as ours trained with just historical price and technical analysis data cannot be guaranteed to work in stock environments with multiple stochastic factors.

CHAPTER 8

Future Work

Experiments in this project were performed on discrete stock price data points using historical stock price data. If we want to deploy the agent in real stock markets, we need to train the agent on real-time data which obviously has high scale and more inherent complexity. The constrains that we added to make agent learn better actually forced it learn slightly sub-optimal rules of the game. Various reinforcement learning algorithms in policy gradient spectrum like Asynchronous advantage actor critic method [31], deep deterministic policy gradients [32] can also be used to train the stock trading agent with adjustable problem setting as per the requirement. Exploring these algorithm might provide some good insight into reinforcement learning for high-entropy time-series data.

Apart from exploring new algorithms, stock price data can be augmented using different other stochastic attributes which influence the price and trend such as news articles, company earning reports to name a few. Simple handcrafted technical analysis features like moving averages, momentum indicator can also be included in the training routine for the agent. The topic of feature selection for stock markets is a strongly debated topic with some researchers suggesting support for addition of such features while some suggest adding handcrafted features in the data adds bias to the training process.

Selecting data to train a generic stock trading agent is still an open issue. Even if we trained a model which generalized fairly well, it is not guaranteed to work on every unseen data and also for any random time-period in the future. Buying

variable number of stocks and selling variable number of stocks for each stock type in the portfolio is extremely complex reinforcement learning decision problem. Scaling current reinforcement learning and deep learning approaches for stock markets is widely researched in academia and industry.

And finally, if such intelligent agent is developed then we would not have properly functioning market.

LIST OF REFERENCES

[1] M. G. Kendall and A. B. Hill, "The analysis of economic time-series-part i: Prices," *Journal of the Royal Statistical Society*, vol. 116, no. 1, pp. 11–34, 1953.

[2] P. H. Cootner, *The random character of stock market prices.* MIT Press, 1964.

[3] B. Malkiel, *A Random Walk Down Wall Street.* W.W. Norton & Company, Inc., 1973.

[4] A. W. Lo and A. C. MacKinlay, "Stock market prices do not follow random walks: Evidence from a simple specification test," *Review of financial studies*, vol. 1, no. 1, pp. 41–66, 1988.

[5] R. Sutton and A. Barto, *Reinforcement learning, an introduction.* MIT Press/Bradford Books, 1998.

[6] N. Prasad and G. Gundersen, "Class Notes: Introduction To Q-learning." https://csml.princeton.edu/sites/csml/files/resource-links/q_learning_notes.pdf, accessed 2019-1-11.

[7] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[8] V. M. et. al, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[9] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 875–889, 2001.

[10] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653–664, 2017.

[11] B. Cheng and D. M. Titterington, "Neural networks: A review from a statistical perspective," *Statistical Science*, vol. 9, no. 1, pp. 2–30, 1994.

[12] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, March 1994.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 21997.

[14] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[15] W. Sharpe, "The sharpe ratio," *The Journal of Portfolio Management*, vol. 21, no. 1, pp. 49–58, 1994.

[16] M. Hargrave, "Definition of Sharpe Ratio." https://www.investopedia.com/terms/s/sharperatio.asp.

[17] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. Holden-Day, 1994.

[18] C. J. Neely, D. E. Rapach, J. Tu, and G. Zhou, "Forecasting the equity risk premium: The role of technical indicators," *Management Science*, vol. 60, no. 7, pp. 1772–1791, 2014.

[19] J. M. Poterba and L. H. Summers, "Mean reversion in stock prices: Evidence and implications," *Journal of Financial Economics*, vol. 22, no. 1, pp. 27–59, 1988.

[20] M. Usmani, S. H. Adil, K. Raza, and S. S. A. Ali, "Stock market prediction using machine learning techniques," *3rd International Conference on Computer and Information Sciences (ICCOINS)*, pp. 322–327, 2016.

[21] A. Sharma, D. Bhuriya, and U. Singh, "Survey of stock market prediction using machine learning approach," *International conference of Electronics, Communication and Aerospace Technology*, pp. 506–509, 2017.

[22] T. Gao, X. Li, Y. Chai, and Y. Tang, "Deep learning with stock indicators and two-dimensional principal component analysis for closing price prediction system," in *IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing*, 2016, pp. 166–169.

[23] M. R. Vargas, C. E. M. dos Anjos, G. L. G. Bichara, and A. G. Evsukoff, "Deep leaming for stock market prediction using technical indicators and financial news articles," in *International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro*, 2018, pp. 1–8.

[24] D. Lu, "Agent inspired trading using recurrent reinforcement learning and lstm neural networks," *Cornell arxiv*, 2017.

[25] Yahoo. "Yahoo finance dataset." [Online]. Available: https://finance.yahoo.com

[26] Quandl. "Financial data api." [Online]. Available: https://www.quandl.com/tools/api

[27] AlphaVantage. "Alpha vantage data api." [Online]. Available: https://www.alphavantage.co

[28] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv e-prints*, p. arXiv:1312.5602, Dec 2013.

[30] M. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation." in *Advances in Neural Information Processing Systems*, 2000, pp. 1057–1063.

[31] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[32] Y. Bengio and Y. LeCun, Eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. [Online]. Available: https://iclr.cc/archive/www/doku.php%3Fid=iclr2016:accepted-main.html